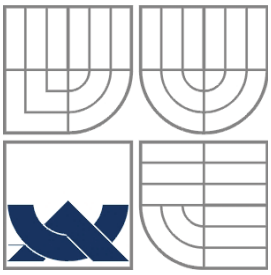
View metadata, citation and similar papers at core.ac.uk

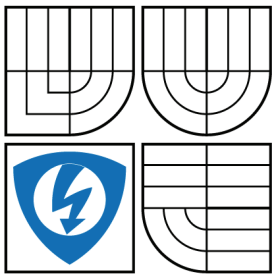
brought to you by CORE

provided by Digital library of Brno University of Technology



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

SW PRO OBSLUHU TESTŮ A GENEROVÁNÍ PROTOKOLŮ

SW FOR CONTROL TESTS AND GENERATING REPORTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

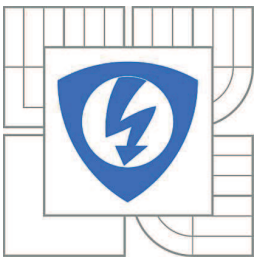
AUTOR PRÁCE
AUTHOR

MICHAL DOČKAL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Lešek Franek

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Michal Dočkal	***ID:***	134471
Ročník:	3	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

SW pro obsluhu testů a generování protokolů

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit program, který zjednoduší práci související s ovládáním a vytvářením samostatných testů určených pro ověřování zařízení pro inteligentní sítě v energetice. Program podle definice testu vyzve uživatele k zadání hodnot, které předá příslušnému testu a po jeho ukončení, na základě vrácených hodnot a šablony, vytvoří automaticky protokol. Program umožní dávkové spouštění testů.

DOPORUČENÁ LITERATURA:

BLANCHETTE, Jasmin a Mark SUMMERFIELD. C GUI programming with Qt 4. Upper Saddle River, NJ: Pearson Hall in association with Trolltech Press, c2006, xvii, 537 p. ISBN 978-013-1872-493.

JANAKA, Ekanayake. Smart Grid Technology and Applications. 2nd ed. Hoboken: John Wiley, 2012. ISBN 11-199-6868-2.

FRANEK, Lešek Data koncentrátor pro chytré sítě: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřící techniky, 2012. 114 s. Vedoucí práce byl Ing. Pavel Kučera, Ph.D.

Termín zadání:	11.2.2013	***Termín odevzdání:***	27.5.2013

Vedoucí práce: Ing. Lešek Franek
Konzultanti bakalářské práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

Abstrakt

V této bakalářské práci je řešen vývoj aplikace sloužící jako grafické prostředí ke spouštění testů a generování protokolů. Nejprve je okrajově rozebrán proces měření a jeho záznam, následně jsou popsány technologie použité při vývoji, dále je navržena vhodná implemetace, a nakonec popsána vytvořená aplikace, včetně nástínění možností dalšího vývoje.

Klíčová slova

Automatické testování, automatické generování protokolů, knihovna Qt, XML, HTML.

Abstract

The content of this work is the development of an application that serves as a graphical interface for running tests and generating reports. At first is marginally analyzed the process of measuring and recording, then the technology used in the development is described. As next, suitable implementations is proposed. Finally createed application is described, including outlining possibilities for further development.

Key words

Automatic testing, automatic generation of reports, Qt framework, XML, HTML.

Bibliografická citace

DOČKAL, M. *SW pro obsluhu testů a generování protokolů .* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2013. 42 s. Vedoucí bakalářské práce Ing. Lešek Franek.

Poděkování

Rád bych poděkoval vedoucímu mé práce Ing. Lešku Franekovi za rady, připomínky a trpělivost, kterou se mnou během vzniku této práce měl. Také bych zde rád poděkoval Mgr. Marii Pospíšilové za pomoc při korektuře práce. V neposlední řadě bych rád poděkoval celé mojí rodině za podporu, kterou mi poskytovala během celého mého studia.

Obsah

1. Úvod

Cílem této práce je vytvořit aplikaci, která bude sloužit jako rozhraní k testům a experimentům. Účelem této aplikace bude zjednodušit práci uživatelům, kteří často na jednom PC provádí více různých testů či experimentů.

Aplikace bude koncipována jako rozhraní mezi uživatelem a konzolovým programem, který provádí experiment. Půjde o formulářovou aplikaci, ve které bude možné načíst šablonu konkrétního testu, zadat jeho parametry a spustit ho. Po skončení této operace program automaticky vygeneruje protokol včetně závěru experimentu. Uživatel bude také moci spustit dávku testů. Program umožňí editaci a uložení načtené šablony, vytvoření nové šablony a načtení starších dat a na jejich základě vygenerovaní nového protokolu.

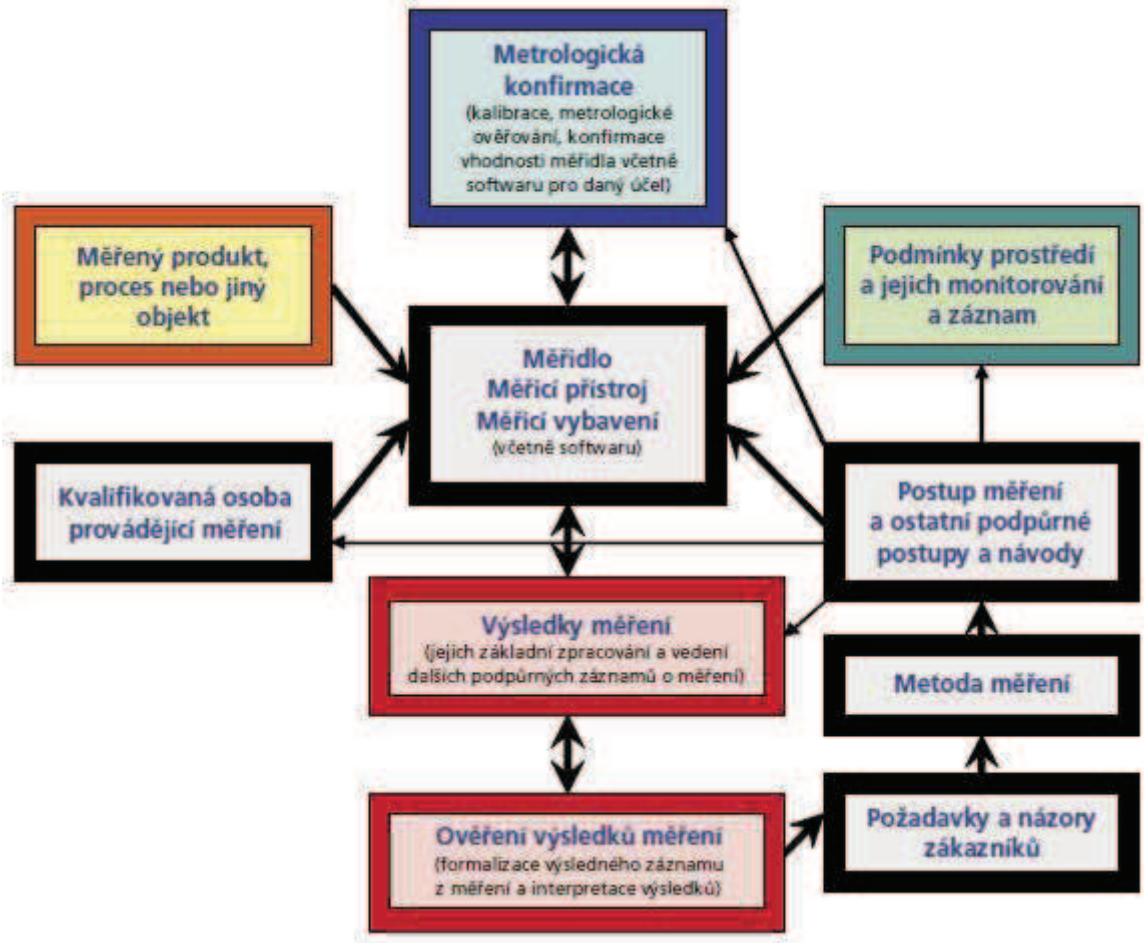
Tato aplikace by měla přinést zjednodušení při provádění automatizovaných testů. Při vytváření nového testu nebude potřeba řešit jeho grafické rozhraní, ani vytváření protokolu. To přinese úsporu času při vytváření takových testů.

V této práci bude postupně rozebrána volba technologií použitých k vytvoření a návrh zmiňované aplikace. Po té bude popsána vytvořená aplikace a rozebrány možnosti dalšího vývoje aplikace.

2. Měření a zpracování výsledků měření

2.1. Charakteristika procesu měření

Proces měření je posloupnost úkonů, které je potřeba vykonat ke stanovení hodnoty měřené veličiny. Jeho schématické znázornění je na obrázku 1.[10]



Obrázek 1: Schematické znázornění procesu měření [10]

Proces měření je tedy posloupnost operací, pomocí kterých dokážeme zjistit hodnotu měřené veličiny. Je také potřeba vzít v úvahu kroky, které zaručí správnost a přesnost naměřených hodnot. Dále je také nutné v procesu měření garantovat vhodnost a smysluplnost záznamu výsledků měření. Je třeba, aby výsledky měření byly interpretovány tak, aby je nebylo těžké pochopit, a aby záznam měření byl vhodný pro zamýšlený účel. Nelze také opomenout fakt, že proces měření neexistuje sám o sobě, ale vzniká na základě potřeb jiných procesů, například produkčních, nebo procesů analýzy výsledků nějakého průzkumu a podobně. [10]

2.2. Záznam měření

Při studiu této tématiky bylo zjištěno, že neexistuje žádná norma či specifikace, která by určovala, jak má vypadat správný záznam měření, neboli protokol. Přesto však existují určitá obecná nepsaná pravidla, která by měla být při vytváření protokolu respektována.

Protokol by měl být dobře čitelný a jednoduše naformátovaný dokumet. Měl by se v něm kdokoli, kdo jej čte, snadno orientovat. Dále by měl obsahovat následující náležitosti [13]:

- Jméno pracovníka či pracovníků, kteří měření prováděli
- Datum měření
- Podmínky, za kterých měření probíhalo, a mohly ovlivnit výsledky měření např. teplota okolního vzduchu, tlak okolního vzduchu a podobně
- Stručný popis problematiky měřeného děje
- Postup měření
- Seznam přístrojů použitých k měření
- Naměřené hodnoty
- Zpracované výsledky měření
- Zhodnocení měření, závěr

V popisu měřeného děje by mělo být stručně uvedeno jaký jev měříme, co je k jeho změření potřeba, jakou veličinu konkrétně měříme a jak z naměřených hodnot zjistíme hodnoty požadované veličiny. Například pokud je měřena teplota pomocí změny odporu materiálu, mělo by být do popisu měření uvedeno, jakým způsobem bude z napětí naměřeného na sondě získána požadovaná hodnota teploty. V postupu by mělo být vysvětleno, jaké kroky budou potřeba ke změření daných hodnot. U zpracování výsledků by po té mělo být uvedeno, jaké hodnoty byly získány výpočtem. Pro každý postup výpočtu musí být uveden alespoň jeden příklad tohoto výpočtu, aby bylo jasné, jak bylo výsledků dosaženo. Dále jsou v části zpracování uváděny různé grafy průběhů změřených veličin. Ty jsou uváděny z toho důvodu, že z nich lze lépe vypozorovat, zda průběh naměřené, či vypočítané veličiny je očekávaného charakteru. V závěru protokolu by vždy mělo být měření zhodnoceno. Měla by zde proběhnout diskuze získaných výsledků a případné zdůvodnění odchylek změřených hodnot od očekávaných výsledků. [13]

Ne vždy je však nutné uvádět všechny tyto náložitosti, záleží vždy na konkrétním měření, na podnikových normách a podobně.

Například při rutinní kontrole napětí na elektrickém vedení by zřejmě nebylo nutné uvádět žádné principy, které se při měření používají, ani by nebylo nutné žádné zpracování výsledků, jelikož bychom měřili napětí přímo. Do závěru bychom poté pouze shrnuli, zda změřené hodnoty vyhovují očekávaným výsledkům, případně jak moc se liší a zda je to problém.

Při laboratorním měření nějakého složitého děje by nicméně bylo vhodné uvést do protokolu vše, včetně popisu problematiky. Při zpracování protokolu by vždy mělo být dodržováno pravidlo, podle kterého musí být ze záznamu schopen měření zopakovat i někdo, kdo se v dané problematice plně neorientuje. [14]

3. Volba technologií vhodných pro vývoj aplikace

V následující kapitole bude diskutována volba nejvhodnějších technologií, které budou použity pro vývoj aplikace.

3.1. Volba IDE

IDE (integrated development environment) je vývojové prostředí. Jedná se o aplikaci, či soubor aplikací, které se používají pro vývoj programů. Na volbu IDE se váže i výběr vhodného programovacího jazyka a také možných knihoven.

3.1.1. Visual Studio

Visual Studio je relativně rozšířené IDE od firmy Microsoft, určené pro vývoj konzolových a formulářových aplikací v programovacich jazycích C, C++, C#, a dalších. Při použití tohoto IDE by bylo vhodné využít právě jazyka C# v kombinaci s knihovnami .NET.

3.1.1.1. Jazyk C#

Jazyk C# je objektový programovací jazyk vyvíjený firmou Microsoft. Na rozdíl od klasického C++, není porgram v jazyce C# kompilátorem překládán do strojového kódu, ale je přeložen do pseodokódu, který je poté spuštěn pomocí CLR (viz. níže). Při spuštění programu je však nutné, aby na počítači, který program spouští, byl nainstalován .NET framework. [1]

3.1.1.2. .NET framework

Aplikace napsané v jazyce C# jsou spouštěny pomocí .NET frameworku. To je vnitřní součást operačního systému Microsoft Windows. .NET obsahuje virtuální spouštěcí mechanismus CLR (common language runtime), který provádí pseodokód, do kterého jsou aplikace v jazyce C# kompilovány. Dále CLR provádí automatické uvolňování již nepoužívané paměti (tzv. garbage collection), řízení vyjímek a správu zdrojů. Také obsahuje sadu knihoven, které poskytují řadu již definovaných tříd, včetně tříd pro práci s řetězci, pro práci se soubory, parsovaní XML, pro vytváření formulářových aplikací a mnoho dalších. [1]

3.1.2. Qt Creator

Qt Creator je multiplatformní IDE vytvořené speciálně pro potřeby vývojářů, kteří využívají knihovnu Qt. Je určen pro vývoj v různých programovacích jazycích. Jedním z nich je jazyk C++, který by bylo vhodné použít v případě volby této knihovny. Jeho součástí je Qt Designer, který slouží k jednoduchému grafickému navrhování

formulářů a Qt Asisstant, který slouží k jednoduchému přístup k dokumentaci Qt přímo z IDE. [2]

3.1.2.1. Knihovna Qt

Qt je multiplatformní knihovnou, kterou lze využít pro tvorbu GUI. Obsahuje také třídy pro zpracování XML, SQL, vláken, či pro práci s grafikou, soubory nebo multimédii. Všechny tyto třídy jsou taktéž multiplatformní. To z ní činí velmi silný nástroj. Díky tomu, že se jedná o open source (knihovna je dostupná zdarma a je vyvýjena komunitou), je velmi rozšířená. [3]

3.1.2.2. Signály a sloty

Signály a *sloty* jsou asi největší předností Qt. Je to způsob komunikace mezi jednotlivými objekty. Každý objekt, který dědí z třídy *QObject* (základní třída knihovny Qt) nebo jejích potomků, může obsahovat signály a sloty. Všechny třídy této knihovny dědí ze třídy *QObject* a tudíž instance těchto tříd mohou mezi sebou tímto způsobem komunikovat. Ve třídách knihovny Qt je velké množství již předdefinovaných signálů a slotů, ale je zde i možnost definování vlastních signálů a slotů.

Celý mechanismus signálů a slotů funguje následujícím způsobem: signály jsou jednotlivými objekty vysílány. Objekt, který jakýmkoliv způsobem změní svůj stav, přičemž jde o změnu, která by mohla být pro okolí zajímavá, vyšle signál a nestará se o to, jestli jej někdo příjme nebo ne.

Objekt, který přijme konkrétní signál, spustí předem definovaný slot. Sloty slouží k reakci na přijatý signál. Je to funkce, která se spustí buď přijetím signálu, nebo ji lze také volat jako klasickou metodu objektu.

Aby mohl objekt na nějaký signál reagovat, je nutné nejprve v tomto objektu spojit konkrétní signál s konkrétním slotem. To se provádí pomocí metody *QObject::connect()*. Jakmile jsou objekty propojeny, vždy se vysláním signálu spustí slot. K jednomu signálu lze připojit více slotů a jeden slot může být připojen k více signálům. [4]

3.1.3. Další možná IDE

Visual Studio a QtCreator samozřejmě nejsou jedinými existujícími IDE. Velmi rozšířené jsou například open source projekty Eclipse či NetBeans. Ty jsou však primárně určeny pro vývoj v jazyce Java. Lze je sice použít i pro vývoj v C++. Avšak pro vývoj v C++ s nimi, na rozdíl od QtCreatoru nebo Visual Studia, není přímo spojena žádná sada knihoven pro tvorbu GUI. [11][12]

3.1.4. Výběr nejvhodnějšího IDE

Pro naše účely je vhodnější využít spíše Qt creator a knihovnu Qt v kombinaci s jazykem C++, než Visual Studio a .NET, hlavně z toho důvodu, že Qt je multiplatformní. Visual Studio lze využít pouze pro vývoj aplikací určených pro operační systém Microsoft Windows. Qt lze navíc, na rozdíl od Visual Studia, užívat zdarma.

3.2. HTML a XML

3.2.1. Původ XML a HTML, jazyk SGML

HTML a XML jsou značkovací jazyky, které vznikly ze staršího značkovacího jazyka SGML (Standard Generalized Markup Language), který byl standardizován v roce 1986. Soubory SGML jsou ukládány jako textové soubory ve znacích ASCII tabulky. To velmi zjednodušuje jejich přenositelnost mezi různými systémy. Přínos SGML spočíval především v možnosti definování struktury informace, která je v souboru uložena. Toto má využití v mnoha oblastech např. v databázích, strukturování textu apod. [5][6]

Informace je v souboru uložena mezi tagy (značkami). Představme si například soubor, který reprezentuje adresář. Ten by mohl mít následující strukturu:

```
<ADRESAR>
        <KONTAKT>
                <JMENO>Jan Novák</JMENO>
                <TELEFON TYP=”mobil”>608782681</TELEFON>
                <ADRESA>
                        <ULICE>Orlí<ULICE>
                        <CP>3<CP>
                        <MESTO>Brno</MESTO>
                </ADRESA>
        </KONTAKT>
        <KONTAKT>
        ⋮
        </KONTAKT>
</ADRESAR>
```

Z tohoto příkladu je vidět jeho snadná zpracovatelnost. Jednotlivé informace jsou rozděleny mezi různé tagy. V tagu *TELEFON* vidíme i příklad takzvaného atributu. V tomto konkrétním příkladě je to atribut *TYP* ve kterém je uložena informace, o jaký typ telefonního čísla se jedná. Pokud by adresář byl pouze textovým souborem, ve kterém by nebyly žádné značky, nemohli bychom jednoduše rozlišit například mezi názvem ulice nebo městem. Obojí se může navíc skládat z několika slov, což by nám ještě více komplikovalo práci.

Každá konkrétní aplikace SGML musí mít svou definici. Ta je označována jako DTD (Document Type Definition). V této definici jsou určeny všechny tagy, jejich

parametry a vztahy mezi jednotlivými tagy. Bývá buď na začátku SGML dokumentu, nebo připojena zvlášť jako soubor *.dtd*. [5]

3.2.2. HTML

HTML je zkratka z anglického HyperText Markup Language (Hypertextový značkovací jazyk). Tento jazyk vznikl na začátku 90. let jako konkrétní aplikace jazyka SGML a slouží k tvorbě a formátování webových dokumentů. Dokumenty v tomto formátu nenesou žádnou strojově zpracovatelnou informaci. Jsou pouhým naformátovaným textem, určeným pro zobrazení v internetovém prohlížeči. Do tohoto textu mohou být vkládány i různé obrázky nebo odkazy na jiné stránky, či je možné naformátovat text do tabulek atd. [7]

Jazyk je stále ve vývoji. Je vyvíjen konsorciem W3C. Poslední verzí jazyka je HTML 5, který je vyvíjen od roku 2007 až do současnosti. Jeho standart plánuje W3C vydat v roce 2014. [8]

HTML bylo zvoleno, protože je velmi rozšířené a známé. Alternativou by mohl být například LaTex, který lze jednoduše převést do velmi rozšířeného formátu PDF. LaTex však není tak rozšířený jako HTML.

K tvorbě protokolu by také bylo možné využít čistých textových souborů. Z hlediska informativní hodnoty vytvořeného protokolu, by tato možnost sice mohla připadat v úvahu, ale textové soubory nelze nijak zvlášť formátovat. Tudíž by takový protokol byl španě čitelný a nebylo by možné do něj vkládat například obrázky, grafy, nebo složitější tabulky. I zde je vidět, že HTML je pro naši aplikaci vhodnější.

3.2.3. XML

XML je zkratkou slov Extensible Markup Language (rozšířitelný značkovací jazyk). Jde o obecný značkovací jazyk, který je v podstatě zjednodušenou verzí SGML. Stejně jako v SGML v něm lze definovat vlastní tagy a je také velmi dobře strojově zpracovatelný. [6]

Stejně jako SGML soubor by XML soubor měl mít vlastní definici v souboru DTD - kvůli jednoduché kontrole jeho validity. DTD souboru XML musí obsahovat následující náležitosti [9]:

- deklaraci elementů (tagů)
- deklaraci atributů
- deklaraci entit
- deklaraci notací

Jako příklad opět použijeme adresář, který jsme navrhli v příkladu v kapitole 3.2.1. Takto by vypadala jeho definice pro XML:

```
<!ELEMENT ADRESAR (KONTAKT+)>
<!ELEMENT KONTAKT (JMENO,TELEFON*,ADRESA?)>
<!ELEMENT JMENO (#PCDATA)>
<!ELEMENT TELEFON (#PCDATA)>
<!ELEMENT ADRESA (ULICE,CP,MESTO)>
<!ELEMENT ULICE (#PCDATA)>
<!ELEMENT CP (#PCDATA)>
<!ELEMENT MESTO (#PCDATA)>

<!ATTLIST TELEFON TYP (mobil | pevna | fax) ”mobil”>
```

Na prvním řádku lze vidět deklaraci elementu *ADRESAR*. Deklarace elementu začíná slovem *!ELEMENT*. Za ním následuje název elementu, v tomto konkrétním případě tedy *ADRESAR*. Na konci deklarace je poté vypsáno, co se v elementu objevuje. Zde je to element *KONTAKT*, přičemž, znak *+* zde znamená, že element *ADRESAR* musí obsahovat alespoň jeden element *KONTAKT*.

Na dalším řádku je deklarován stejným způsobem element *KONTAKT*. Je zde řečeno, že se v něm může objevovat element *JMENO*, *TELEFON* a *ADRESA*. Za elementem *JMENO* není uveden žádný symbol. To znamená, že v elementu *KONTAKT* vždy musí být právě jeden element *JMENO*. Za elementem *TELEFON* je symbol *. Tím je určeno, že element *TELEFON* se v elementu *KONTAKT* může objevovat vícekrát, ale nemusí se objevit vůbec. Posledním elementem, který se zde může objevit, je element *ADRESA*. Symbol ? říká, že element *ADRESA* se v kontaktu může vyskytovat právě jednou nebo vůbec.

Další je deklarace elementu *JMENO*. Slovo *#PCDATA* říká, že element *JMENO* neobsahuje žádný další element, ale obsahuje konkretní data (ta jsou ve formě textu). Stejně tak i element *TELEFON* obsahuje data. Element *ADRESA* obsahuje elementy *ULICE*, *CP* a *MESTO*, všechny právě jednou a všechny obsahují data.

Na posledním řádku je deklarace atributu *TYP*, který se nachází v tagu *TELEFON* a může obsahovat text *„mobil“*, *„pevna“*, nebo *„fax“*. Pokud není tento atribut uvedený, pak je výchozím textem *„mobil“*.

Pokud by bylo potřeba, aby byl tag nebo atribut prázdný, bylo by použito slovo *EMPTY*, naopak, pokud by bylo jedno, co se v tagu bude nacházet, bylo by použito slovo *ANY*. [9]

3.2.4. Výhody XML a alternativní technologie

XML je vhodné pro svou dobrou strojovou zpracovatelnost. Alternativou k XML by mohl být JSON (JavaScript Object Notation). JSON je používán zejména v kombinaci se skriptovacím jazykem JavaScript. Knihovana Qt, která bude využita k tvorbě grafického rozhraní, JSON sice podporuje, avšak XML je rozšířenější než JSON, díky čemuž je XML

podporováno větším množstvým parserů než JSON. Jelikož použití jazyka JSON není nutné, použijeme raději XML.

Další alternativou by mohl být opět textový soubor. To by však nebylo příliš vhodné, protože čistý text není moc dobře strojově zpracovatelný. Pokud bychom v textu začali vytvářet nějaké vlastní speciální značky, za účelem rozlišení dat v souboru, museli bychom napsat kód pro jejich zpracování. Proto je určitě vhodnější použít XML, které, jak již bylo řečeno, je přímo podporováno v Qt. Navíc je formát XML standardizovaný a rozšířený.

4. Návrh aplikace

V následující kapitole budou rozebrány návrhy dílčích částí vytvářené aplikace. Nejprve bude navrhnut objektový model aplikace. Následně budou rozebrány návrhy DTD souborů XML používaných pro komunikaci a návrh nejvhodnějšího formátu vzorového protokolu. Poté budou pomocí vývojových diagramů popsány hlavní funkce aplikace. Nakonec bude navrženo vhodné grafické rozhraní aplikace.

4.1. Návrh objektového modelu

Program bude postaven na třech základních třídách, kromě kterých budeme samozřejmě využívat třídy z knihovny Qt. První z těchto tříd se bude starat o GUI aplikace. Další třída bude spravovat lineární seznam, ve kterém budou uloženy jednotlivé parametry testu. Poslední třída bude definovat podobu prvků v seznamu, ve kterých budou uložena data jednotlivých parametrů testu, včetně jejich omezení, výchozích hodnot, datových typů a samotných odesílaných dat.

4.1.1. Třída hlavního okna aplikce

Tato třída bude definovat GUI celé aplikace. Bude potomkem třídy *QMainWindow*, což je třída z knihovny Qt, která je určena k vykreslování hlavního okna aplikace.

Tato třída se bude starat o veškeré operace prováděné nad oknem aplikace - jako je vykreslování jednotlivých menu a povolování či zakazování kliknutí na položky v těchto nabídkách obsažených, dále pak vykreslování jednotlivých módů aplikace, překreslování rolovací lišty, rolování rolovací lištou apod.

Bude obsahovat ukazatele na jednotlivé grafické objekty hlavního okna aplikace a také ukazatel na třídu lineárního seznamu.

4.1.2. Třída lineárního seznamu

V této třídě definujeme metody pro práci s lineárním seznamem. Také bude obsahovat metody potřebné pro načítání a vytváření XML souborů pro ukládání a načítání šablon a metody potřebné k vytváření a čtení XML souborů pro komunikaci s programem provádějícím test. Jelikož bude výhodné použít mechanizmu signálů slotů z knihovny Qt pro komunikaci mezi jednotlivými objekty aplikace, bude tato třída dědit ze třídy *QObject*, aby tento mechanizmus mohla používat.

Tato třída bude obsahovat ukazatel na první prvek v lineárním seznamu. Dále pak ukazatel na objekt hlavního okna aplikace. Tento ukazatel je nutné, protože musí být předáván grafickým prvkům, které budou vytvářeny pro každou položku lineárního seznamu zvlášť, jako různá pole pro zadání hodnoty parametru, pole pro zadání minima

a maxima parametru a podobně. Bude předáván každému novému prvku, vždy při jeho vytvoření. Jednotlivé grafické prvky z knihovny Qt totiž potřebují znát adresu prvku, ve kterém se mají vytvořit - v našem případě tedy adresu hlavního okna apliakce.

Také bude potřeba zde do vhodných proměnných ukládat cestu k programu, který bude provádět experiment a cestu ke vzorovému protokolu, ze kterého bude vytvářen protokol po proběhnutí testu. Bylo by vhodné v této třídě ukládat i cestu k souboru, ze kterého aplikace načetla šablonu, aby v případě ukládání změněné šablony byla ušetřena práce uživateli a ten nemusel pracně dohledávat místo na disku, odkud soubor s šablonou načetl.

4.1.3. Třída prvku lineárního seznamu

Instance této třídy budou jednotlivými prvky v lineárním seznamu. Vyskytuje se zde však problém - bude zapotřebí do jednoho seznamu ulkládat data o různých datových typech (int, real, bool a string). Při podrobnější analýze problému však zjistíme, že data načítáme z XML souboru v podobě textových řetězců, a že není zapotřebí s nimi provádět žádné složitější operace, než jejich vkládání do vstupních polí, jejich přepis hodnotami ze vstupních polí a jejich porovnávání. Data tedy můžeme bez problémů nechat v seznamu uložená jako textové řetězce. V případě, že bude potřeba data mezi sebou nějakým způsobem porovnávat, převedeme si je na dané datové typy, jejich hodnoty si uložíme do pomocných proměnných daných datových typů, a následně data porovnáme.

Při vytváření datových struktur jako je lineární seznam se většinou dbá na to, aby navržené struktury byly univerzální, to znamená, aby výslednou strukturu bylo možné použít pro uložení různých typů dat. Například do jedné struktury bychom ukládali celá čísla a do další textové řetězce. Do seznamu, který bude použit, však budou ukládána vždy data stejného datového typu, tudíž není zapotřebí tento problém řešit.

V této třídě bude zapotřebí znalost ukazatele na hlavní okno aplikace, abychom mohli vytvářet další grafické objekty, což bude řešeno, jak již bylo zmíněno výše, předáváním tohoto ukazatele při vytváření nového prvku. Také by bylo vhodné, aby v této třídě byla definována proměnná, ve které by bylo uloženo pořadí instance této třídy v lineárním seznamu. Dalšími proměnnými budou textové řetězce do kterých budou ukládána data načtená z šablony experimentu. Nakonec také musí obsahovat ukazatele, do kterých budeme ukládat adresy jednotlivých grafických prvků, jako například vstupní pole pro změny načtených dat, vstupní pole pro zadání odesílané hodnoty parametru a podobně. Právě kvůli těmto objektům bude potřeba, aby zde byl uložen i ukazatel na hlavní okno aplikace.

I v této třídě bude vhodné využít signály a sloty pro komunikaci s ostatními objekty aplikace.

4.2. Návrh DTD pro požívané XML soubory

Program bude používat tři různé definice souborů XML - jednak pro komunikaci s programem, který provádí test a také k ukládání šablon. Všechny tyto DTD byly navrženy přímo pro navrhovanou aplikaci. Dále pak bude pro každý test nutné aby uživatel nadefinoval vzorový protokol testu způsobem, který bude popsán níže.

4.2.1. Šablona testu

Šablony jednotlivých experimentů budou uloženy v souboru XML, pro který byla navržena tato definice:

```
<!ELEMENT parametry (cesta, protokol, promenna*)>
<!ELEMENT cesta (#PCDATA)>
<!ELEMENT protokol (#PCDATA)>
<!ELEMENT promenna (nazev, typ, vychozi, vyraz, min, max)>
<!ELEMENT nazev (#PCDATA)>
<!ELEMENT typ (#PCDATA)>
<!ELEMENT vychozi (#PCDATA)>
<!ELEMENT vyraz (#PCDATA)>
<!ELEMENT min (#PCDATA)>
<!ELEMENT max (#PCDATA)>
```

Celý soubor začíná elementem *parametry*. Uvnitř tohoto elementu se nachází právě jeden element *cesta*, uvnitř kterého je uložena cesta k programu provádějícímu experiment. Dále je uvnitř elementu *parametry* vždy jeden element protokol, ve kterém je uložena cesta ke vzorovému protokolu experimentu. Posledním možným elementem, který se může vyskytovat v elementu parametry, je element *promenna*. Těchto elementů se v elementu *parametry* může vyskytovat libovolné množství. V elementech *promenna* jsou uloženy všechny elementy zastupující vlastnosti jednotlivých parametrů experimentu. Jsou to elemety *název, typ, vychozi, vyraz, min* a *max*, ve kterých je uložen název parametru, jeho datový typ, jeho výchozí hodnota, regulární výraz, který musí odesílaný parametr splňovat v případě, že jde o řetězec, a minimální a maximální hodnota, které nesmí odesílaný parametr přesáhnout v případě, že jde o číselný typ.

4.2.2. Odesílaná data

Pro data odesílaná k testu byl navržen soubor XML s tímto DTD:

```
<!ELEMENT test (promenna*)>
<!ELEMENT promenna (#PCDATA)>
```

Soubor s odesílanými daty bude začínat elementem *test,* uvnitř kterého bude předem nespecifikovaný počet elementů *promenna*. Uvnitř těchto *elementů* budou uloženy hodnoty jednotlivých parametrů odesílaných programu, provádějícímu

experiment. Ten by měl vědět, v jakém pořadí jsou tyto parametry v souboru seřazené a co znamenají.

4.2.3. Vrácení dat

Poté, co experiment skončí, přepíše program, který jej provádí, získaná data do souboru XML podle DTD, které bylo navrženo speciálně pro tento účel:

```
<!ELEMENT vysledky (hodnota*)>
<!ELEMENT hodnota (#PCDATA)>

<!ATTLIST hodnota nazev (#PCDATA)>
<!ATTLIST hodnota vyhovuje (true | false)>
```

Celý soubor je uzavřen v elementu *vysledky*. Uvnitř tohoto elementu jsou uloženy vrácené hodnoty a to jednotlivě, v elementech *hodnota.* Navíc tento element musí vždy obsahovat dva atributy. Prvním je atribut *nazev*, ve kterém je uložen název vrácené hodnoty, díky kterému aplikace bude vědět, kam má tuto hodnotu uložit ve výsledném protokolu. Dále element *hodnota* musí obsahovat atribut *vyhovuje*. Tento atribut může nabývat dvou hodnot, buď *true* a nebo *false*, a značí, zda výsledná hodnota odpovídá omezením daného testu.

4.2.4. Návrh formátu vzorového protokolu

Vzorový protokol nebude klasickým XML souborem, ale půjde o HTML soubor, do kterého budou, do předem daných míst, vloženy nepárové tagy *hodnota,* s parametrem *nazev.* V tomo parametru bude zadán název proměnné, která má být do toho místa vložena. V místě, do kterého si bude uživatel přát vypsat automatický závěr, bude vložen nepárový tag *zaver*. Aplikace po provedení testu tento vzorový protokol načte a zkopíruje jej na místo, keré určí uživatel - s tím rozdílem, že tagy *hodnota* nahradí hodnotou vrácenou testem, která bude mít název shodný s názvem hodnoty uloženým v atributu *nazev* daného tagu *hodnota*. Tag *zaver* aplikace nahradí automaticky vygenerovaným závěrem.

4.3. Návrh funkčnosti programu

Aplikaci si z grafického hlediska rozdělíme na dvě základní obrazovky. Nazveme je mód spouštění testů a mód editace šabloby. Jak z jejich názvů vyplývá, první z obrazovek bude sloužit k nastavování hodnot parametrů a spouštění testů, druhá obrazovka pak bude sloužit k editaci šablon jednotlivých testů. V následujícím textu budou navrženy algoritmy, které bude uživatel v těchto dvou obrazovkách spouštět volbami jednotlivých položek z menu.

4.3.1. Vytvoření nové šablony

Při vytváření nové šablony bude třeba provést operace uvedené ve vývojovém diagramu na obrázku 2. Nejprve tedy bude nutné vyprázdnit lineární seznam a smazat všechna ostatní data, která nejsou uložena v seznamu. Tato data se totiž při vytváření nové šablony stávají neplatnými. Poté bude okno překresleno do módu editace. To provede objekt, který bude mít na starosti lineární seznam, protože v lineárním seznamu budou uloženy jednotlivé pramatry testu a grafické prvky jejich úpravy. Následně objekt, vykreslující hlavní okno, překreslí menu do požadované podoby a upravý rolovací lištu tak, aby vyhovovala nové velikosti zobrazovaných dat.

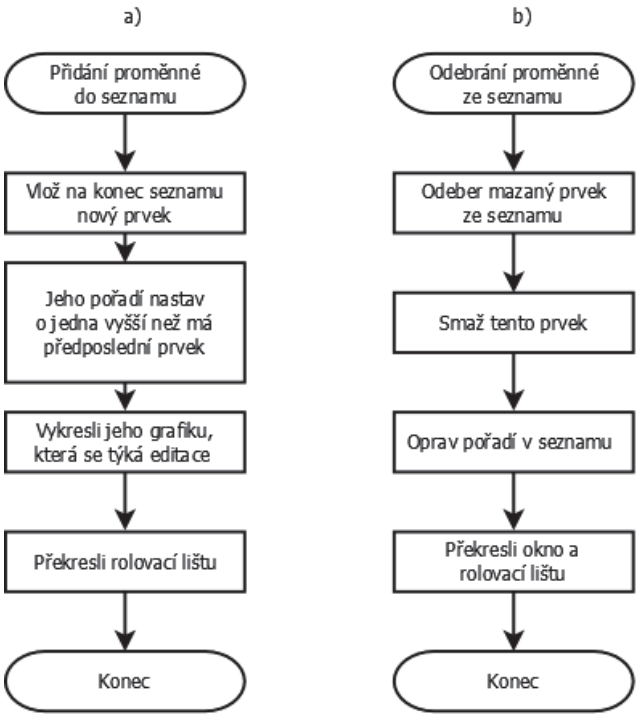
Obrázek 2: Vývojový diagram vytváření nové šablony

4.3.2. Spuštění editace šablony

Tento proces bude možné spustit pouze z módu spouštění testů. Operace překreslí okno do módu editace a to stejným způsobem jako v předchozím případě, s tím rozdílem, že nebude provádět žádné mazání. Touto volbou se bude uživatel pouze přepínat do módu úpravy šablony, tudíž žádná data mazat nebude potřeba. Provede se tedy pouze překreslení okna, menu a rolovací lišty tak, jak bylo popsáno výše.

4.3.3. Přidání a odebrání proměnné ze šablony

Oba tyto procesy bude možné spustit pouze z módu editace šablony. Při přidávání nové proměnné se vytvoří nový prvek, který se vloží na konec seznamu a jeho pořadí se nastaví o jedna vyšší, než má předchozí prvek. Poté se vykreslí grafika daného prvku a překreslí se rolovací lišta tak, aby odpovídala novým rozměrům okna. Vývojový diagram procesu je zobrazen na obrázku 3a.



Obrázek 3: Vývojový diagram a) přidání a b) odebrání proměnné ze seznamu

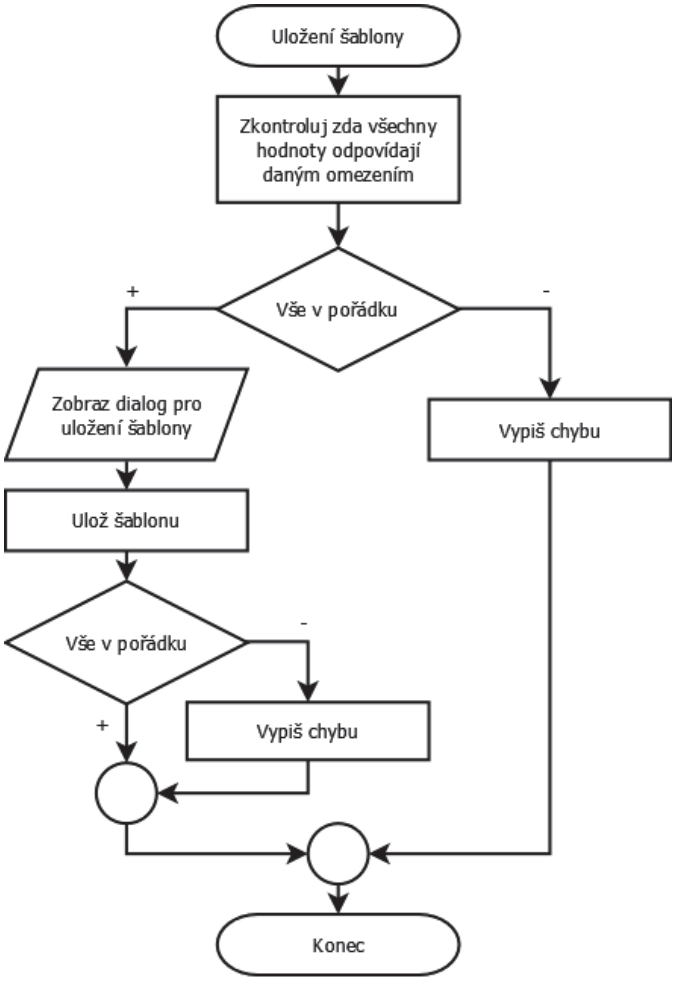
Při mazání se nejprve daný prvek vyjme ze seznamu. Poté bude smazán. Následně bude potřeba projít celý seznam a zkontrolovat, zda mají všechny prvky uloženo správné pořadí. Z tohoto pořadí totiž budeme počítat umístění jednotlivých grafických prvků v okně aplikace. Nakonec bude třeba překreslit okno, aby po vymazaném prvku nezůstala mezera a také bude třeba překreslit rolovací lištu. Vývojový diagram mazání můžeme vidět na obrázku 3b.

4.3.4. Uložení šablony

Tento proces bude možné spusti pouze z módu editace. Při ukládání šablony bude nutné nejprve provést kontrolu dat ve všech vstupních polích. Aplikace zkontroluje, zda ve vstupních polích pro zadávání omezení jsou honoty přípustné pro daný datový typ. U čísel pak bude třeba zkontrolovat, zda omezení minimální hodnoty čísla není větší než omezení maximální hodnoty čísla. Pro řetězce bude nutné zkontrolovat, zda se v poli pro zadání regulárního výrazu opravdu vyskytuje regulární výraz. Poté bude nutné zkontrolovat, zda zadané výchozí hodnoty u jednotlivých parametrů odpovídají zadaným datovým typům parametrů. Nakonec proběhne kontrola, zda zadané výchozí hodnoty odpovídají všem zadaným omezením.

Pokud se během jakékoliv z těchto kontrol zjistí, že hodnota ve vstupním poli neodpovídá omezením, aplikace na to uživatele upozorní chybovým hlášením a operace se ukončí. Pokud všechny hodnoty budou vyhovovat, aplikace zobrazí dialog pro uložení šablony a uživatel vybere soubor, do kterého chce šablonu testu uložit. Aplikace poté tento soubor otevře a pokusí se do něj nahrát data uložená v seznamu. Pokud během

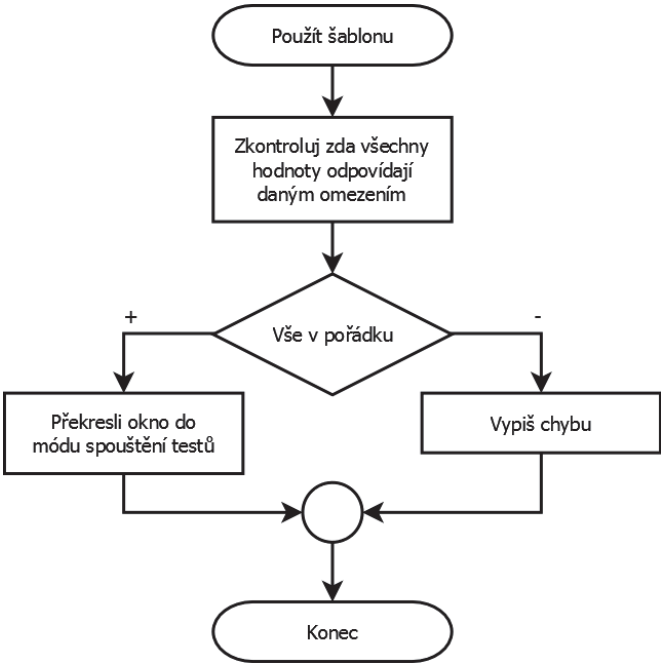
procesu ukládání dojde k chybě, aplikace o tom vypíše chybové hlášení a proces se ukončí. V případě, že uložení souboru proběhne v pořádku, proces také skončí, ale bez vypsání jakéhokoli dalšího hlášení. Vývojový diagram tohoto procesu je zobrazen na obrázku 4.



Obrázek 4: Vývojový diagram uložení šablony

4.3.5. Přepnutí zpět do módu spouštění testů

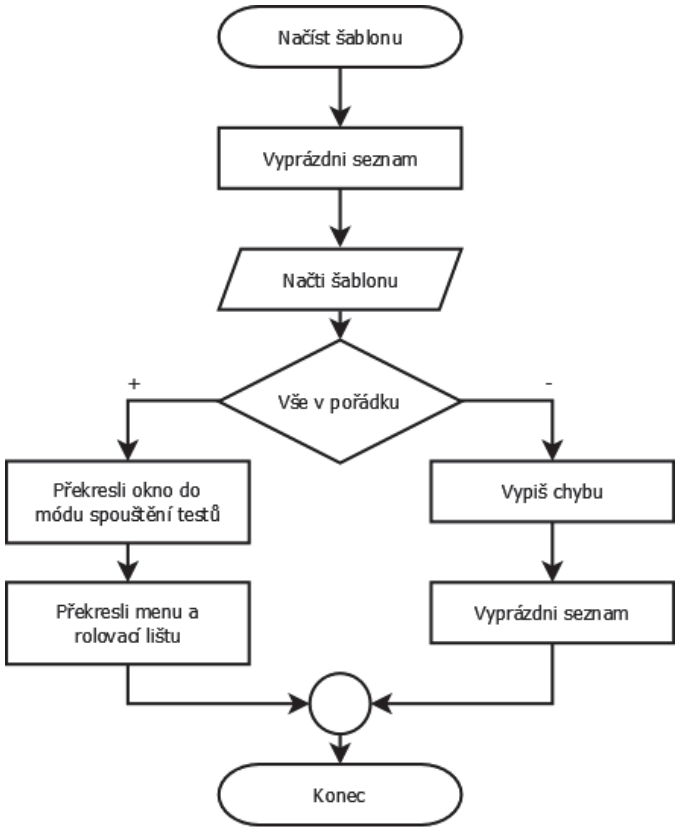
Tento proces bude možné spustit pouze pokud se uživatel bude nacházet v módu editace šablony. Po spuštění tohoto procesu proběhne stejná kontrola vstupních polí, jako v případě ukládání šablony, to znamená, že se zkontrolují vstupní pole - zda odpovídají datovým typům a omezením. Pokud ne, vypíše se chybové hlášení a operace se ukončí. Pokud je vše v pořádku, proces skryje všechny grafické prvky patřící do módu editace a zobrazí výpis vstupních polí pro zadání hodnot jednotlivých parametrů před odesláním k testu. Algoritmus je znázorněn ve vývojovém diagramu na obrázku 5.



Obrázek 5: Vývojový diagram přepnutí do módu spouštění testů

4.3.6. Načtení šablony ze souboru

Při načítání šablony bude nejprve nutné vyprázdnit seznam, aby do něj mohla být načtena nová data. Poté proces uživateli zobrazí dialog pro otevření souboru XML s šablonou testu, uživatel vybere soubor a aplikace se jej pokusí otevřít.



Obrázek 6: Načtení šablony k testu

Jakmile je soubor otevřen aplikace, se z něj pokusí načíst data. Pokud vše proběhne v pořádku vykreslí se vstupní pole pro zadaní parametrů testu a zapíší se do nich načtené výchozí hodnoty. Poté se překreslí menu a rolovací lišta. Pokud se během načítaní souboru vyskytne chyba, aplikace zobrazí chybové hlášení a vyprázdní seznam, aby v něm po neúspěšném načítání nezůstala nějaká nevalidní data. Celý proces je zobrazen na obrázku 6.

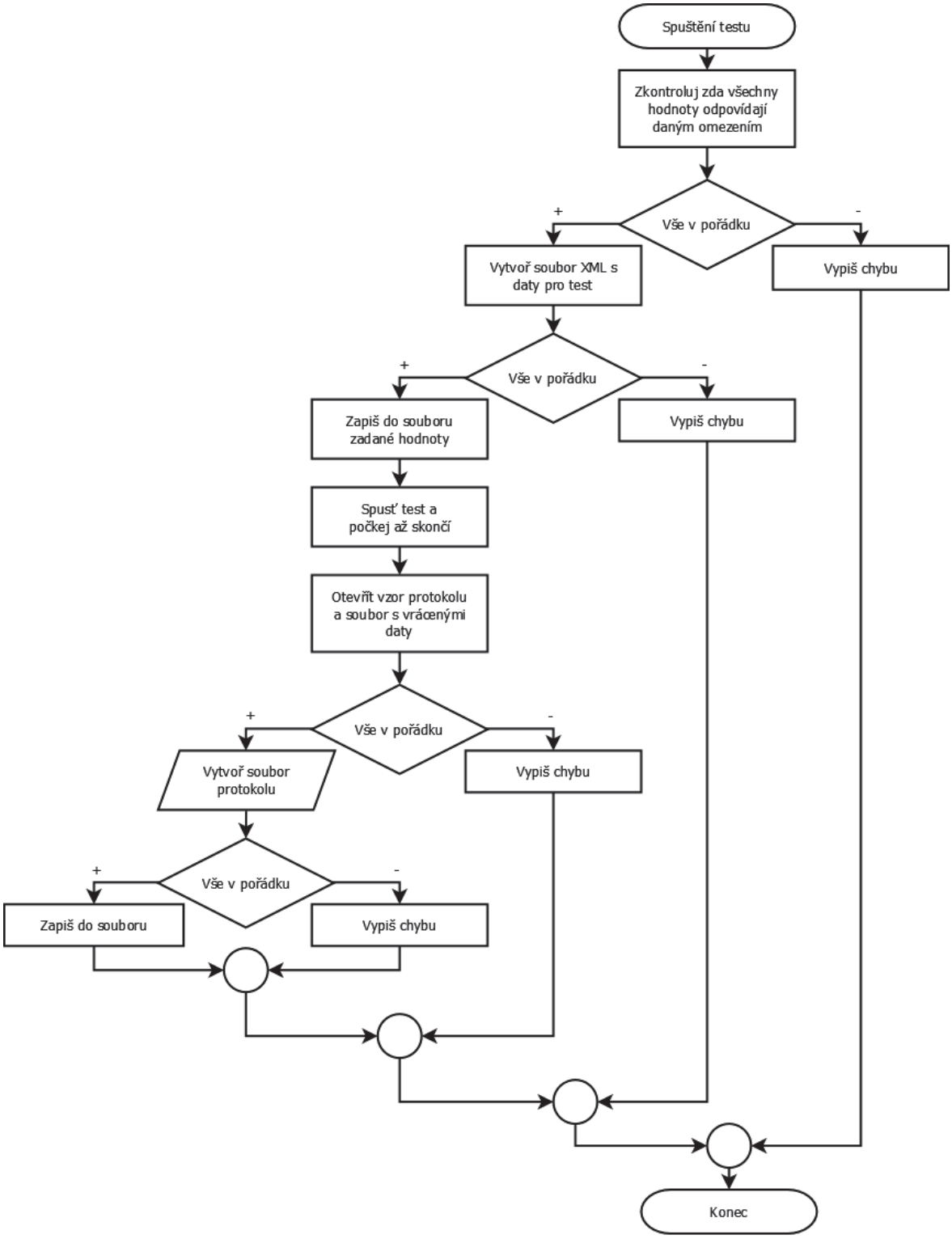
4.3.7. Spuštění testu a vytvoření protokolu

Na obrázku 7 je zobrazen vývojový diagram procesu spuštění testu. Tento proces bude možné spustit pouze pokud se uživatel bude nacházet v módu spuštění testu. Po spuštění projde proces všechna vstupní pole pro zadávání hodnot parametrů odesílaných k testu a zkontroluje, zda hodnoty v těchto polích odpovídají svým datovým typům, a zda hodnoty splňují omezení daných parametrů.

Pokud nějaká z hodnot nebude splňovat jakoukoli z podmínek, vypíše se o tom chybové hlášení a proces skončí. Pokud bude vše v pořádku, pokusí se proces, v adresáři ve kterém je uložena aplikace, která bude provádět test, vytvořit XML soubor pro odeslání hodnot testu. Pokud se nepodaří tento soubor vytvořit, vypíše o tom aplikace chybové hlášení a proces se ukončí. Pokud vše proběhne v pořádku, zapíše proces do souboru hodnoty jednotlivých parametrů a soubor uzavře.

Poté bude spuštěna aplikace, která bude mít za úkol provést test, přičemž jako prarametr jí bude předána cesta k souboru, ve kterém budou uložena data potřebná k provedení testu. Aplikace tento soubor přečte a na základě hodnot jednotlivých parametrů provede test. Poté tato aplikace otevře XML soubor, ve kterém jí byla zaslána data potřebná k provedení testu. Obsah tohoto souboru vymaže a nahradí jej výsledky experimentu, přičemž zkontroluje, zda výsledky odpovídají očekávaným hodnotám, případně nějakým jiným omezením. Pokud nějaký z výsledků testu nebude odpovídat omezením, do souboru tuto skutečnost zaznamená. Následně XML soubor uzavře a ukončí se.

Mezitím aplikace, která je předmětem této práce, bude čekat na ukončení aplikace provádějící test. Po ukončení aplikace provádějící test, se aplikace, která je předmětem této práce, pokusí otevřít soubor s vrácenými daty a načíst ho. Také se pokusí otevřít soubor se vzorovým protokolem. Pokud během některé z předcházejících operací dojde k chybě, aplikace vypíše chybové hlášení a proces se ukončí. Bude-li vše v pořádku, zobrazí se uživateli dialog pro zadání cesty k souboru, do kterého bude uložen protokol měření. Uživatel vybere soubor a aplikace se jej pokusí otevřít. Pokud se soubor nepodaří otevřít, aplikace vypíše chybu a proces skončí.



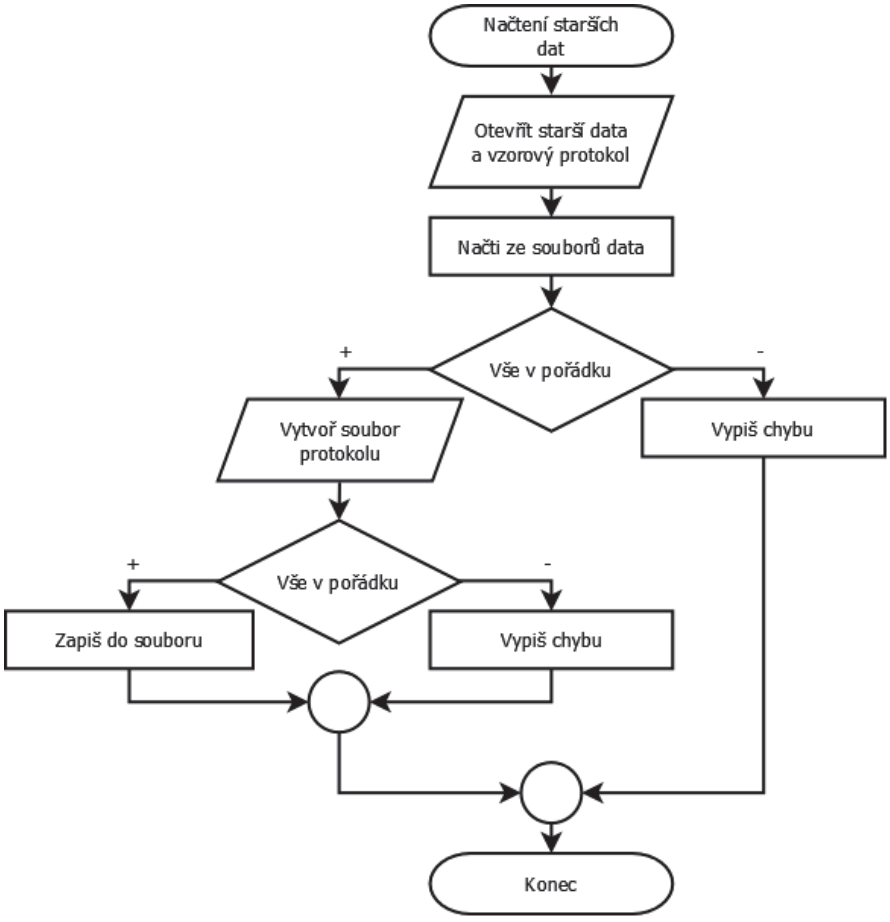
Obrázek 7: Vývojový diagram procesu spuštění testu

Pokud bude všechno v pořádku, aplikace začne postupně pročítat a kopírovat soubor vzorového protokolu do uživatelem zvoleného souboru. Jakmile narazí na místo, kde by měla být zapsána hodnota měření, nahradí toto místo danou hodnotou načtenou z XML souboru s výsledky měření. Pokud aplikace narazí na místo, do kterého by měl být vložen závěr, projde všechny hodnoty. Budou-li všechny hodnoty vyhovat, zapíše

do protokolu informaci o tom, že měření proběhlo v pořádku. Pokud jedna nebo více hodnot nebude vyhovovat, aplikace do protokolu vypíše informaci o tom, že výsledky měření jsou mimo vyhovující hodnoty a všechny nevyhovující hodnoty vypíše. Po přečtení a zkopírování celého souboru aplikace uzavře všechny soubory, se kterými pracuje a proces skončí.

4.3.8. Načtení dat staršího testu a vytvoření protokolu

Tato operace bude probíhat obdobně jako předchozí oprace. Rozdílem však bude, že v tomto případě nebudou odesílána žádná data a nebude spouštěn test. Uživateli se na začátku zobrazí dialog pro výběr XML souboru s daty vrácenými testem. Následně se uživateli zobrazí dialog pro výběr souboru vzorového porotokolu. Jakmile uživatel tyto soubory vybere, aplikace se oba soubory pokusí otevřít a načíst ze souboru XML výsledky měření.



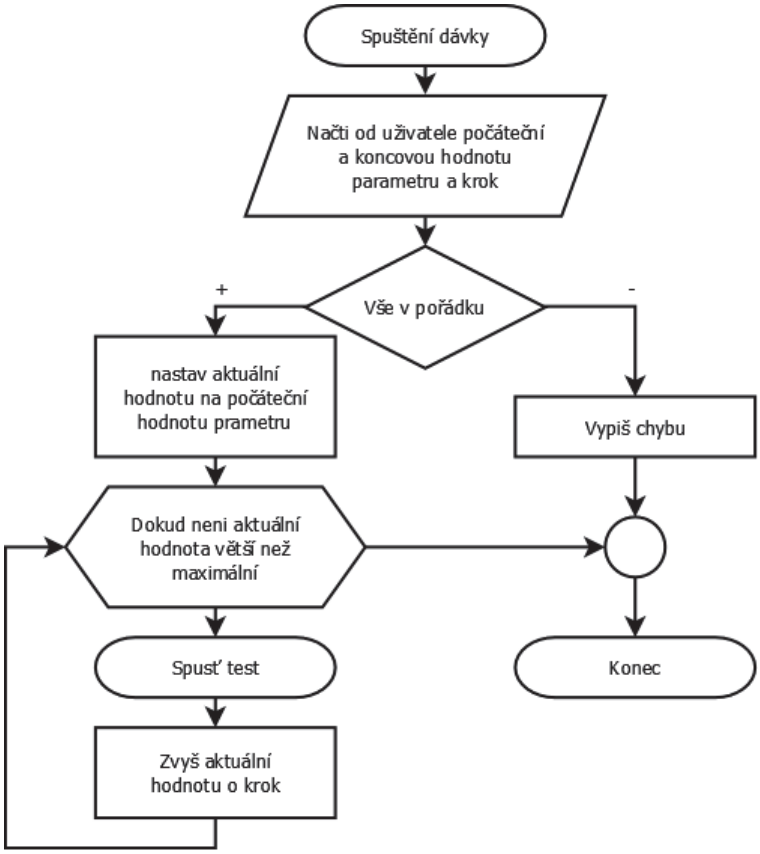
Obrázek 8: Vývojový diagram procesu načtení starších dat a vytvoření protokolu

Pokud během některé z předcházejících operací dojde k chybě, aplikace o tom uživatele informuje chybovým hlášením a proces skončí. Proběhne-li vše bez problémů, bude uživateli zobrazen dialog pro uložení souboru protokolu. Uživatel vybere vhodný soubor a aplikace se jej pokusí otevřít.

Pokud se to aplikaci nepodaří, vypíše o této skutečnosti chybové hlášení a proces se ukončí. Pokud proběhne otevření souboru pro vytvoření protokolu bez problému, aplikace do něj začne zapisovat protokol, stejným způsobem, jako v případě vytváření protokolu přímo po skončení testu. Vývojový diagram tohoto procesu je na obrázku 8.

4.3.9. Spuštění dávky testů

Tento proces bude možné spustit pouze v případě, že se bude uživatel nacházet v módu spuštění testu. V menu si uživatel vybere parametr, pro který bude chtít spustit dávku testů. Ve výběru nebude možné zvolit parametr datového typu bool, nebo string, protože pro ně nemá smysl spouštět dávku. Po zvolení prametru se spustí proces zobrazený na vývojovém diagramu z obrázku 9.



Obrázek 9: Vývojový diagram spuštění dávky testů

Po spuštění tohoto procesu bude uživatel vyzván, aby zadal počáteční hodnotu zvoleného parametru, krok, o který se bude hodnota při každem dalším experimentu zvyšovat a konečnou hodnotu, při jejímž dosažení se dávka ukončí. Aplikace zkontroluje zda, zadané hodnoty vyhovují datovému typu daného parametru, a zda vyhovují omezením daného parametru.

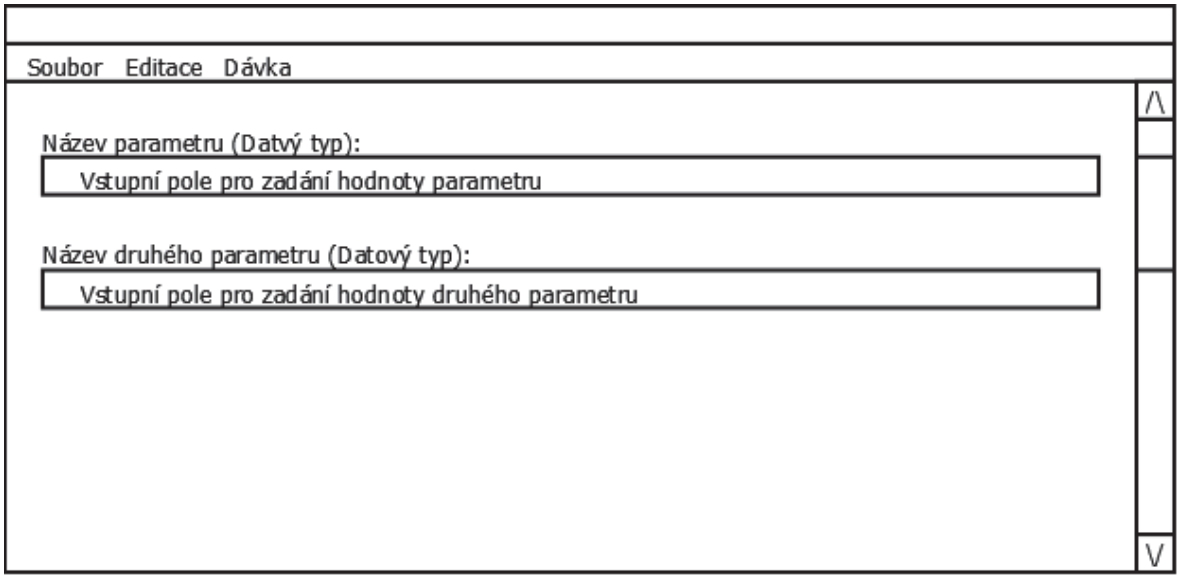
Pokud nevyhovují, aplikace tuto skutečnost uživateli zobrazí pomocí chybového hlášení. Pokud je vše v pořádku, nastaví se hodnota parametru, pro který spouštíme dávku, na uživatelem zadanou počáteční hodotu. Poté se v cyklu vždy spustí proces

spuštění testu, který byl popsán v kapitole 4.3.7. Následně se k hodnotě parametru, pro který je dávka spouštěna, přičte krok a spouštění testu se opakuje, dokud hodnota parametru není větší než maximální hodnata. V tu chvíli se proces ukončí.

4.4. Návrh GUI

Grafické rozhraní aplikace bude tvořit okno formuláře, které bude mít na pravé straně rolovací lištu a v horní části menu, které bude rozděleno na tři části: soubor, editace a dávka. V části soubor bude uživatel provádět základní operace - jako vytvoření šablony, načtení šablony, spuštění testu a podobně. V menu editace bude moci uživatel přidávat do šablony parametry, ukládat změněnou šablonu atd. V posledním menu, které bude možné otevřít pouze pokud se aplikace bude nacházet v módu spuštění testů, bude výpis všech parametrů, přičemž volbou jednoho z nich se pro něj spustí dávka testů.

Obsah okna se bude lišit v závislosti na tom, zda se aplikace bude nacházet módu spouštění testů nebo v módu editace šablony. V módu spouštění testů, jehož návrh je na obrázku 10, budou vstupní pole pro zadávání hodnot jednotlivých paramaterů. Pokud v menu uživatel vybere možnost spuštění testu, hodnoty zadané v těchto polích aplikace zkontroluje a odešle v XML souboru aplikaci provádějící test.

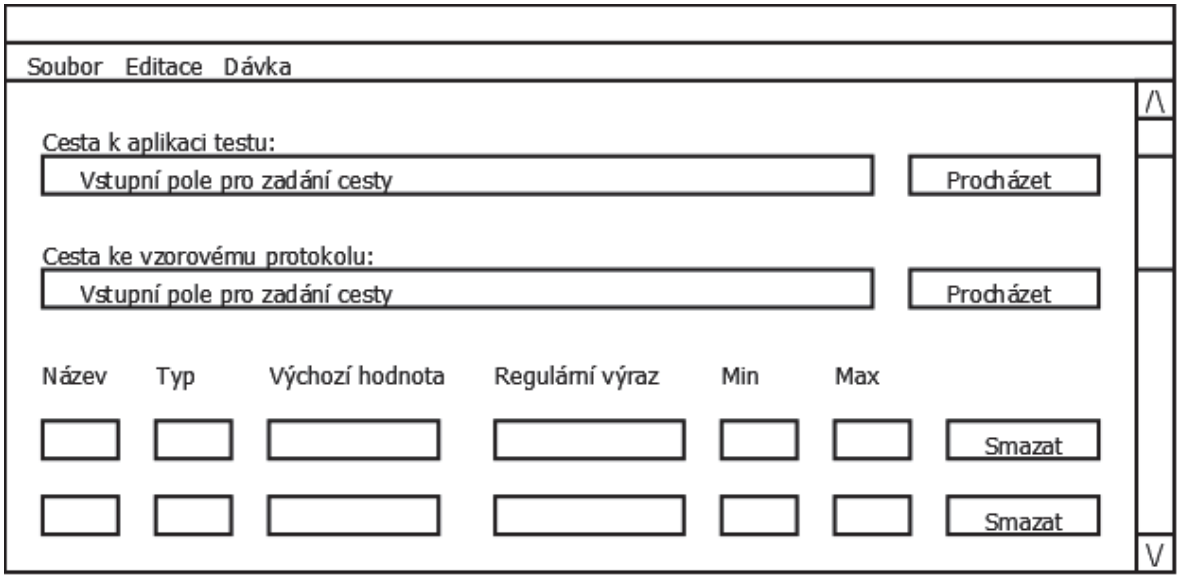


Obrázek 10: Návrh obrazovky módu spouštění testů

Na obrázku 11 je návrh okna módu editace šablony. V tomto okně bude možné editovat načtenou šablonu. V horní části budou dvě vstupní pole pro zadání cesty k aplikaci testu a cesty ke vzorovému protokolu testu. Vedle každého z nich bude tlačítko procházet, které otevře dialog k výběru daného souboru.

Pod těmito dvěma vstupními poli bude tabulka načtených parametrů. V této tabulce budou vstupní pole, ve kterých budou načteny vlastnosti prametrů (v každém řádku jeden parametr). Uživatel je zde může změnit a přes menu *editace* uložit. Vedle

vstupních polí každého parametru se budou nacházet tlačítka pro smazání daného parametru. Při kliknutí na kterékoliv z těchto tlačítek aplikace ze šablony odebere parametr, na jehož tlačítko uživatel klikl. Pokud bude aplikace na této obrazovce, nebude možné otevřít menu pro spouštění dávek.



Obrázek 11: Návrh obrazovky módu editace

5. Popis vytvořené aplikace

Základní funkcí aplikace, která byla navržena v předchozím textu, je poskytnout jednotné grafické uživatelské rozhraní pro spouštění testů a experimentů, přičemž konkrétnímu experimentu odešle data, na základě předem definované šablony k testu. Aplikace byla navržena jako formulářová aplikace a byla vytvořena pomocí Qt Creatoru, v jazyce C++, s použitím tříd knihovny Qt.

Po spuštění je okno aplikace prázdné (až na menu v horní liště a rolovací lištu) a uživatel má několik možností. Může načíst šablonu testu pomocí záložky *Soubor → Načíst Šablonu* v horní liště programu. Pokud uživatel zvolí tuto možnost, aplikace zobrazí dialog pro výběr souboru a poté se okno aplikace překreslí do módu spouštění testů. Dále uživatel může vytvořit novou šablonu pomocí záložky Soubor → *Nová šablona.* V tomto případě se okno překreslí do módu editace šablony a aplikace smaže všechna data, která do ní byla načtena během jakékoli předchozí práce. Uživatel také může načíst data z nějakého již proběhlého experimentu a na jejich základě nechat program vytvořit protokol o měření a to pomocí záložky *Soubor → Načíst data a vytvořit protokol*. V tomto případě se okno nijak nepřekreslí, ale zobrazí se dialogy pro načtení souboru s daty, načtení vzorového protokolu a uložení souboru protokolu. Uživatel může také aplikaci ukončit přes záložku *Soubor → Ukončit.*

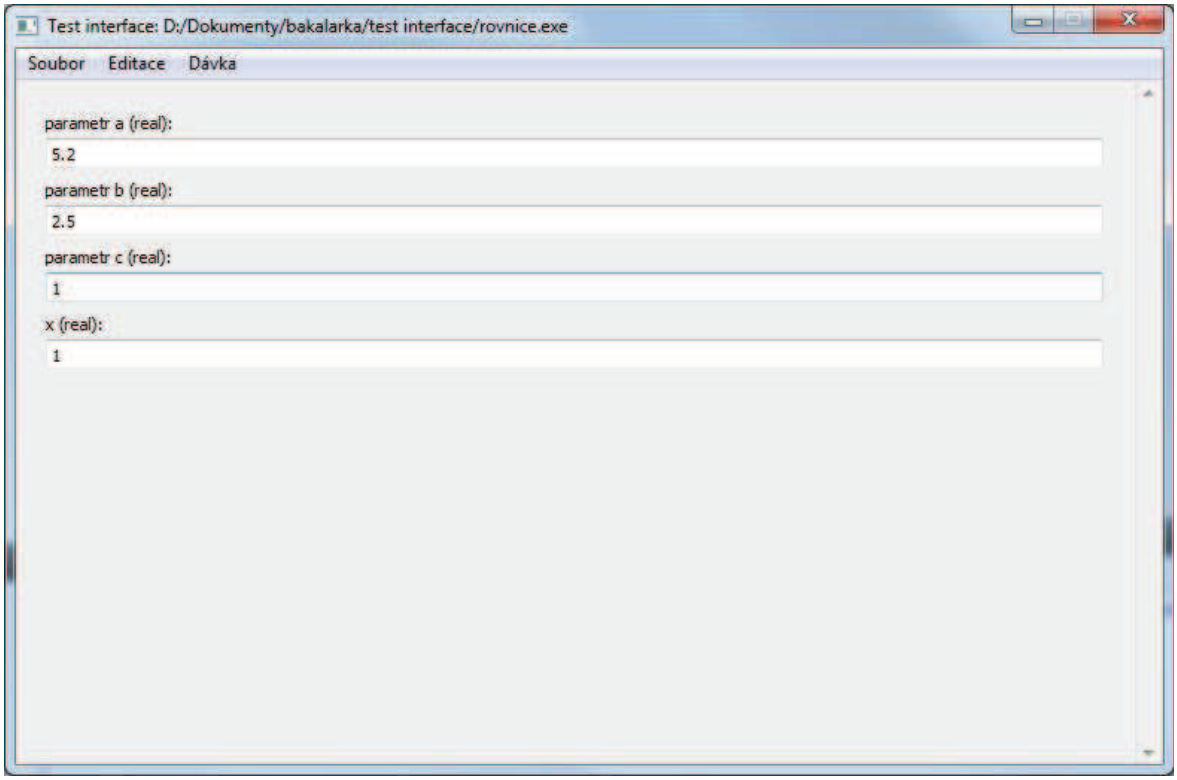
5.1. Mód spouštění testů

Při překreslení do módu spouštění testů (obrázek 12) se v okně zobrazí všechny načtené parametry následujícím způsobem - vždy se zobrazí vstupní pole, ve kterém je načtena výchozí hodnata daného parametru, nad tímto vstupním polem je vždy popisek se jménem prametru a jeho datovým typem.

V menu se při překreslení do tohoto módu aktivují všechny možnosti v záložce soubor. Jsou to všechny záložky, které byly popsány výše v textu. Jejich funkčnost zůstává stejná. Navíc se aktivuje záložka *Soubor → Spustit.* Pokud uživatel vybere tuto možnost, aplikace načte hodnoty zadané ve vstupních polích a odešle je k provedení testu.

Dále se do menu *Dávka* vykreslí všechny parametry, přičemž aktivní jsou pouze ty, které jsou datového typu int, nebo real. Pro typ bool, nebo string dávky nejsou spouštěny. Výběrem jednoho z povolených parametrů se zobrzí dialog pro zadání počáteční hodnoty. Po jejím zadání se spustí dialog pro zadání kroku a nakonec dialog pro zadání konečné hodnoty. Poté proces načte hodnoty ze vstupních polí ostatních parametrů a postupně spouští testy, přičemž modifikuje hodnotu parametru, pro který je dávka spuštěna.

Poslední akcí, která se při vykreslování tohoto módu stane, je deaktivování všech položek v menu *Editace*, až na položku *Editace → Editovat šablonu.* Touto záložkou se uživatel přepne do módu editace šablony.



Obrázek 12:Mód spouštění testů

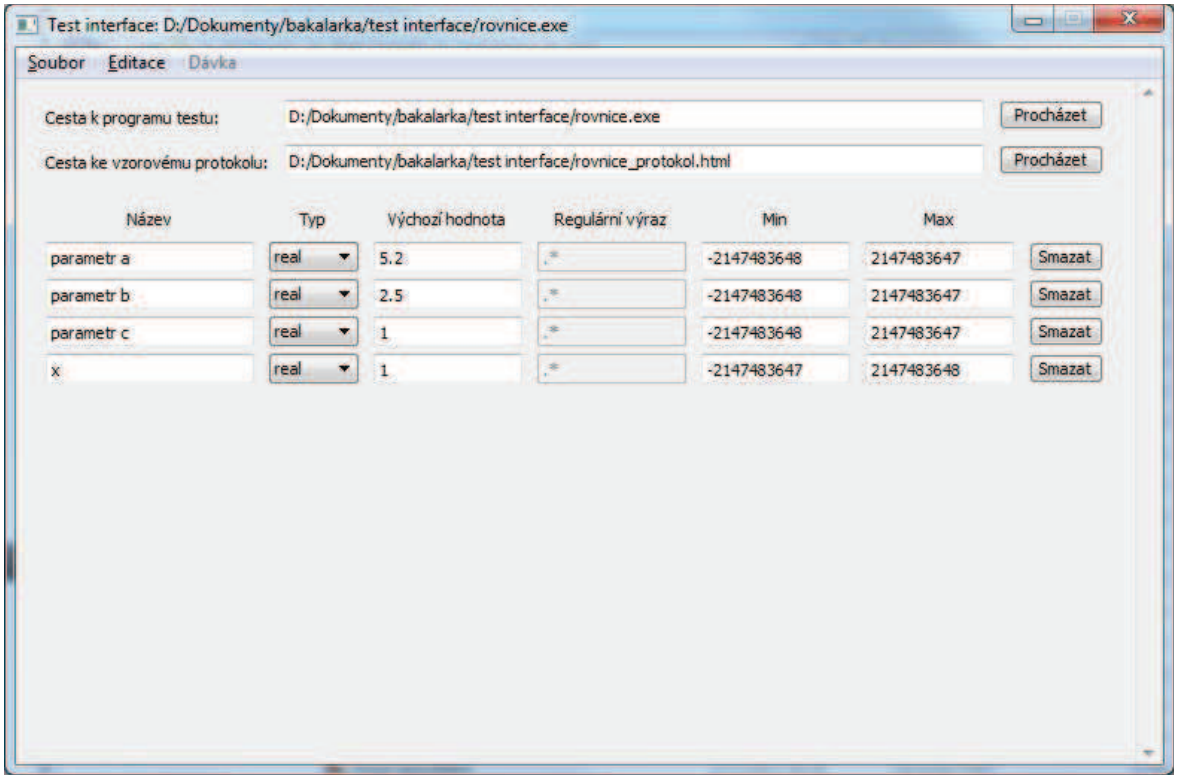
5.2. Editace šablony

Když je aplikace přepnuta do módu editace, hlavní okno se překreslí do podoby, která je na obrázku 13. V horní části okna se vykreslí dvě vstupní pole pro zadání cesty k aplikaci provádějící test a cesty k souboru vzorového protokolu. Vedle obou těchto polí jsou tlačítka, pomocí kterých se spustí dialog pro otevření souboru. Při kliknutí na horní tlačítko uživatel vybere aplikaci testu, při kliknutí na spodní tlačítko vybere soubor vzorového protokolu.

Pod těmito vstupními poli je tabulka s vlastnostmi jednotlivých parametrů. V tabulce je pro každý parametr vždy vstupní pole pro zadání názvu, dále výběrové pole (ComboBox) pro vybrání datového typu parametru, vstupní pole pro zadání výchozí hodnoty parmetru (tato hodnota se u všech parametrů vypíše ve vstupních polích pro zadání hodnoty odesílané k testu, při přepnutí do módu spouštění testů), vstupní pole pro zadání regulárního výrazu, který musí parametr splňovat a vstupní pole pro minimální a maximální hodnotu, které může parametr nabývat. Hodnoty zadané ve třech posledních vstupních polích však pro některé datové typy nemají žádný smysl. Konkrétně má smysl určovat regulární výraz pouze pro text (datový typ string). Proto u všech ostatních možných datových typů (int, real, bool) je toto pole neaktivní a nelze do něj zapisovat.

Dále pak minimální a maximální hodnota má smysl pouze pro celá a reálná čísla (int, real). U datových typů string a real jsou tato pole neaktivní. Při změně datového typu parametru se vždy daná pole buď aktivují nebo deaktivují podle toho, jaký je nový datový typ parametru.

Posledním prvkem v tabulce je tlačítko *Smazat*, které slouží k odstranění daného parametru ze šablony. Pokud na něj uživatel klikne, parametr, vedle kterého je tlačítko umístěno bude smazán a okno se překreslí tak, aby po smazaném parametru nezůstalo v okně prázdné místo.



Obrázek 13: Mód editace šablony

V módu editace šablony je menu *dávka* neaktivní, protože nelze spouštět dávku pro nějaký parametr, pokud nejsou definovány hodnoty ostatních parametrů. Bylo by sice možné při spuštění dávky u ostatních parametrů využít jejich výchozích hodnot, to by však mohlo být matoucí. Například by si uživatel mohl otevřít nějakou šablonu a v módu spuštění testu zadat hodnoty k odeslání testu. Potom by zjistil, že mu nevyhovuje nějaké omezení v šabloně a přepnul by se do módu editace, ve kterém by si upravil jeden parametr a v menu by viděl, že lze spustit dávku. Dávku by spustil. Programu testu by však byla odeslána jiná data, než jaká zadal v módu spouštění testů, což by si uživatel nemusel uvědomit. Proto je vhodnější spouštění dávek v módu editace šablony deaktivovat.

V menu *Editace* jsou aktivní všechny možnosti až na možnost *Editace → Editovat šablonu,* která je ze zřejmého důvodu neaktivní. V menu editace se uživatel již nachází a aktivní možnost přepnout se znovu do módu editace by opět mohla být matoucí. Aktivní je ale možnost *Editace → Použít šablonu*, která okno překreslí zpět do módu spouštění

testů, přičemž do vstupních polí pro zadání hodnot k odeslání testu zapíše výchozí hodnoty jednotlivých parametrů. Dále je aktivní možnost *Editace → Uložit šablonu*, která zobrazí dialog pro výběr souboru, do kterého má být šablona uložena a následně šablonu do užvatelem vybraného souboru uloží. Poslední možností je *Editace → Přidat parametr.* Při výběru této možnosti se na konec šablony vloží nový parametr, defaultně typu int, avšak tento datový typ lze jednoduše změnit pomocí výběrového pole pro výběr datového typu daného parametru.

V menu *Soubor* jsou aktivní všechny záložky, kromě záložky *Soubor→Spustit test.* Ta je neaktivní a to ze stejného důvodu, kvůli kterému není vhodné z módu editace spouštět dávku testů. Ostatní záložky *Soubor → Nová šablona, Soubor → Načíst šablonu, Soubor → Načíst data a vytvořit protokol* a *Soubor → Ukončit,* jsou aktivní. Tyto záložky mají stejnou funkci jako v případě, kdy program spustíme, a nenacházíme se ani v jednom z módů zobrazení.

6. Další vývoj aplikace

6.1. Objektový model

Při dalším vývoji by bylo vhodné upravit objektový model aplikce. V současném stavu aplikace bez problémů funguje, avšak kód je nepřehledný. Ve třídě, která se má starat o lineární seznam je totiž zároveň řešena i funkčnost aplikace. Bylo by tedy vhodné přidat do modelu další třídu, do které by byly přesunuty metody, které se starají o načítání souborů, ukládání souborů, spouštní dávek a testů atd. V této třídě by pak byla vytvořena instance třídy zastřešující lineární seznam. Tím by z kódu pro zpracování souborů zmizely části, ve kterých procházíme jednotlivé prvky seznamu a byly by nahrazeny pouhým voláním metod ze třídy lineárního seznamu.

Dalším možným řešením by bylo odebrat ze třídy, která se stará o lineární seznam, všechny akce související s lineárním seznamem, odebrat z prvku seznamu ukazatele na další a předchozí prvek a použít některou z datových struktur implementovaných přímo knihovnou Qt. Do zvolené struktury by poté byly vkládány jednotlivé prvky seznamu. Stejně jako v předchozím případě by tím bylo odděleno načítání a ukládání do souborů od ukládání a čtení dat ze seznamu.

6.2. Vlákna

Další vhodnou úpravou by bylo rozdělení procesů grafického rozhraní a procesů lineárního seznamu, načítání a ukládání dat a spouštění testu a čekání na jeho ukončení do dvou různých vláken programu.

V současném stavu dojde při provádění nějakého delšího testu k „zamrznutí“ grafického rozhraní, což není moc vhodné. Uživatel totiž nemůže vědět že „zamrznutí“ aplikace je způsobeno záměrně. Aplikace čeká na ukončení testu aby mohla načíst jím vrácená data a vytvořit protokol.

Řešením by tedy bylo provést výše zmíněné rozdělení procesů do různých vláken. V případě spuštění testu by okno aplikace nezamrzlo. Bylo by ale dobré při spuštění testu uzamknout všechny ovládací prvky okna aplikace. Jinak by mohlo dojít k tomu, že uživatel během provádění testu změní nebo smaže nějaká data, bez kterých aplikace nebude schopná dokončit proces vytvoření protokolu. V případě dávky by pak také při změně dat mohlo dojít k přerušení dávky. Také by bylo v tom případě vhodné přidat nějaký grafický prvek, který by ukazoval průběh testu či dávky. Jinak by mohlo dojít k situaci, že uživatel spustí dávku nějakých déle trvajících testů a zůstane mu otevřené okno apikace, ve kterém jsou zablokovány všechny akce, přičemž uživatel neví, jak dlouho bude tento stav trvat.

6.3. Protokol

6.3.1. Parametry testu

Při vytváření protokolu by mohlo být přínosné, kdyby do protokolu bylo možné vkládat i parametry odesílané k testu. Většinou je totiž potřeba do protokolu zapsat jak výsledky měření, tak i hodnoty parametrů, ze kterých měření vycházelo. Například - pokud měříme úbytek napětí na odporu, určitě je vhodné do protokolu zanést jak zjištěný úbytek napětí, tak hodnotu napětí, které bylo nastaveno na zdroji.

V současné podobě to musíme řešit tak, že program, který provádí test, musí parametry, které chceme zapsat do protokolu posílat zpět, i když je aplikace pro zpracování protokolů zná.

Možným řešením by bylo přidat do vzorového protokolu další tag. V současné době jsou zde tagy *hodnota* a *zaver*. Byl by použit například nepárový tag *parametr* s atributem *nazev*, přičemž by byl princip stejný jako u tagu *hodnota.* Při procházení vzorového protokolu by aplikace ve chvíli, kdy narazí na tag *parametr*, tento nahradila hodnotou parametru se jménem, které by se shodovalo s textem v atributu *nazev*.

6.3.2. Tabulky v protokolu

Dále by bylo dobré nějakým způsobem řešit i formátování dat do tabulek přímo v aplikaci provádějící test. V současném stavu musí být celá tabulka napsána ručně ve vzorovém protokolu, musí do ní být ručně zapsány i názvy jednotlivých hodnot, které budou do tabulky vkládány, což by v případě složitějších měření mohlo být velmi pracné. Určitě také existují měření, u kterých je velikost tabulek nutné měnit při každém měření, což současný model neumožňuje.

V tomto případě by mohlo řešením být přepracování definice XML pro návratová data takovým způsobem, aby v tomto souboru mohly být definovány tabulky. Aplikace by si poté tabulku načetla a ve vhodném místě by ji celou vložila do protokolu. Vhodné místo by mohlo být ve vzorovém protokolu definováno dalším tagem, například *tabulka*, který by obsahoval opět parametr *nazev*, díky kterému by bylo možné určit, kterou tabulku vykreslit v tomto konkrétním místě.

6.3.3. Grafy v protokolu

Dalším vhodným rozšířením automatické tvorby protokolu by mohla být automatická tvorba grafů z dat vrácených ve formě tabulky. Toto by mohlo fungovat následujícím způsobem. Aplikace testu by odeslala data v XML souboru. Konkrétní data, která by měla být vykreslena v grafu, by byla naformátována v tabulce. Ve vzorovém protokolu by byl vložen další tag, například *graf*, s parametrem *nazev* a parametry určujícími, která konkrétní data mají být v grafu na ose x, a která na ose y. Určitě by bylo

možné navrhnout i další aributy, které by například určovaly rozsah hodnot na jednotlivých osách, typ očekávaného průběhu, barvu vykreslovaného průběhu a podobně.

V případě, že by aplikace narazila na tag *graf* v souboru vzorového protokolu, přečetla by všechny jeho atributy. Poté by prohledala data vrácená experimentem, zda se mezi nimi nenachází tabulka se stejným názvem, jako byl v atributu *nazev* daného tagu *graf*. Z nalezené tabulky by poté načetla data, která by měla být zobrazena v grafu a podle atributů z tagu *graf* by tento graf vytvořila a naformátovala. Následně by aplikace vytvořila soubor obrázku (nejvhodnější by byl asi GIF nebo PNG, pro možnost průhledného pozadí) do kterého by vytvořený graf uložila. Nakonec by celý tag *graf* byl nahrazen právě vytvořeným obrázkem grafu.

Navržené řešení by bylo výhodné zejména tím, že by tabulka, ze které byl graf vytvořen, mohla být vložena do protokolu i ve formě tabulky a ne jen jako graf. Z toho vyplývá, že by nebylo nutné stejná data odesílat z experimentu vícekrát, jednou jako tabulku a podruhé jako graf. Bylo by také možné jeden graf v jednom protokolu zobrazit vícekrát, ale různě naformátovaný. Například jednou celý graf a podruhé jen část hodnot ale s větším rozlišením a tím zvýraznit například nějaké kritické místo.

7. Závěr

Cílem této práce bylo vytvořit aplikaci, která umožní spouštět různé testy a experimenty ve formě konzolových aplikací z jediné aplikace, přičemž test či experiment vrátí nějaká data, která aplikace zpracuje do protokolu. To by přineslo zjednodušení práce programátorům, kteří vytvářejí testovací aplikace. Hlavní úsporou je nejspíše čas, který by bylo nutné věnovat tvorbě grafického rozhraní takové aplikace.

Zmiňovaná aplikace byla vytvořena. Obsahuje navíc rozhraní pro editaci a vytváření šablon spouštění jednotlivých testů. Komunikace mezi aplikací a aplikací provádějící experiment probíhá pomocí souborů ve formátu XML. Protokoly jsou vytvářeny na základě načtení vzorového protokolu ze souboru formátu HTML, ve kterém jsou vloženy speciální značky pomocí kterých, aplikace rozezná, jaké informace má kam vložit. Je zde zpracován i automatický závěr protokolu. Následně je vytvořený protokol uložen do souboru HTML, který si zvolí uživatel.

V této práci byl nejprve velmi okrajově popsán proces měření. Také byly popsány náležitosti, které by měl obsahovat dobrý záznam o měření.

V další kapitole byla diskutována volba nejvhodnějších technologií pro tvorbu zadané aplikace. Nejprve bylo za nejvýhodnější IDE pro tvorbu aplikace vybrán Qt Creator. Následně byly rozebrány značkovací jazyky HTML a XML a bylo vysvětleno, proč jsou pro naši aplikaci vhodné.

Další kapitola se zabývala návrhem této aplikace. Byl navržen objektový model, na kterém by bylo nejvhodnější aplikaci vytvořit. Také byly navrženy formáty souborů pro ukládání šablon testů a komunikaci aplikace s aplikací provádějící experiment. Velká část této kapitoly pak byla věnována návrhu procesů uvnitř aplikace. Posledním tématem této kapitoly byl návrh grafického rozhraní.

V předposlední kapitole pak byla popsána vytvořená aplikace a to především z hlediska dvou hlavních obrazovek do kterých, byla rozdělena. Byl také popsán význam jednotlivých záložek v menu.

V poslední kapitole byly navrženy konkrétní úpravy, které by bylo vhodné provést při dalším vývoji aplikace. První úpravou je nutnost změny objektového modelu, který se po vytvoření aplikace ukázal být nepřehledným. Dále je nutné přidat rozdělení procesů grafického rozhraní a ostatních procesů do různých vláken. V současném stavu totiž aplikace během provádění testu zamrzne. To sice není funkční problém, ale není to příliš uživatelsky přívětivé.

Velký prostor k dalšímu vývoji má aplikace především v automatické tvorbě protokolu. Určitě by bylo vhodné tento proces rozšířit o možnost vkládat do protokolu přímo parametry odesílané k testu. Bylo by možné i definovat možnost odesílání dat

strukturovaných do tabulek, což by uživateli usnadnilo práci při vytváření nových testů. Nebylo by totiž nutné tabulky ve vzorovém protokolu vytvářet ručně. Navíc by toto rozšíření zlepšilo schopnost aplikace nějak konkrétně data formátovat a tím například možnost automaticky generovat do protokolu grafy.

8. Použitá literatura a zdroje

[1] Introduction to the C# Language and the .NET Framework. MICROSOFT. *Visual Studio* [online]. 2013 [cit. 2013-05-15]. Dostupné z: http://msdn.microsoft.com/en-us/library/vstudio/z1zx9t92.aspx

[2] Qt Creator. *Qt Project* [online]. 2011 [cit. 2013-01-04]. Dostupné z: http://qt-project.org/wiki/Category:Tools::QtCreator

[3] Qt (framework). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012-12-23 [cit. 2012-12-30]. Dostupné z: http://en.wikipedia.org/wiki/Qt_(framework)

[4] Signals & Slots. In: *Qt Project* [online]. Oslo, Norway, 2011 [cit. 2012-12-30]. Dostupné z: http://qt-project.org/doc/qt-4.8/signalsandslots.html

[5] SGML: Standard Generalized Markup Language. KOSEK, Jiří. Vše o WWW: Vše co jste kdy chtěli vědět ale báli jste se zeptat [online]. 1999 [cit. 2013-01-01]. Dostupné z: http://www.kosek.cz/clanky/cw/sgml.html

[6] XML: staronový formát. KOSEK, Jiří. Vše o WWW: Vše co jste kdy chtěli vědět, ale báli jste se zeptat[online]. 1999 [cit. 2013-01-01]. Dostupné z: http://www.kosek.cz/clanky/xml/xml-historie.html

[7] HTML. *Tvorba-webu.cz* [online]. 2003-2008 [cit. 2013-01-01]. Dostupné z: http://www.tvorba-webu.cz/html/

[8] HTML 5 Definition complete, W3C moves to interoperability testing and performance. In: *W3C* [online]. 2012-12-17. [cit. 2013-01-02]. Dostupné z: http://www.w3.org/

[9] DTD: Definice typu dokumentu pod lupou. KOSEK, Jiří. *Vše o WWW: Vše, co jste kdy chtěli vědět, ale báli jste se zeptat* [online]. 1999 [cit. 2013-01-02]. Dostupné z: http://www.kosek.cz/clanky/xml/xml-01.html

[10] LUDVÍK, Vladimír. *Sborníky technické harmonizace: systém managemetu měření*. Praha: Q-Art, 2004, 90 s. Dostupné z: http://www.unmz.cz/

[11] *NetBeans IDE: The Smarter and Faster Way to Code* [online]. 2013 [cit. 2013-05-25]. Dostupné z: https://netbeans.org/

[12] *Eclipse: The Eclipse Foundation open source comunity website* [online]. 2013 [cit. 2013-05-25]. Dostupné z: http://www.eclipse.org/

[13] HAVLÍKOVÁ, M. FEKT VUT. *Vyhláška pro laboratorní cvičení - kurs BMVE*. Brno, 2012. Dostupné z: https://www.vutbr.cz/

[14] FEKT VUT. *Měření fyzykálních veličin: Zpracování protokolů*. Brno, 2013. Dostupné z: https://www.vutbr.cz/

9. Seznam obrázků